Android Gradle 详细讲解

文档变更记录：

变更日期	变更原因	记录人
2018 年 9 月 19 日星期三	创建	LC
2019 年 4 月 22 日星期一	添加解决 Android Studio 文件同步错误的办法	LC

文档说明：

鉴于 Android Gradle 规则比较多，网络上资料比较分散，Android 应用的构建完全依赖于 Gradle，遇到问题时常束手无措，遂整理出此文档以供参考。

目录

1. 前言

1.1. Gradle 简介

Gradle 本身提供一些基本的概念和整体核心的框架，其他用于描述真实使用场景逻辑的都以插件扩展的方式来实现。比如构建 Java 应用，就是通过 Gradle 内置的 Java 插件来实现的，AndroidGradle 插件就是基于其内置 Java 插件实现的。

把插件应用到项目中，会帮助你在项目的构建过程中做很多事情，比如：测试、编译、打包；可以添加以来配置到项目中；也可以对项目进行一些约定，比如应用 Java 插件之后，约定 src/main/java 目录下是我们源代码存放的位置，在编译的时候也是编译这个目录下的 Java 源文件。我们只需要安装插件约定的方式，使用其提供的任务、方法或扩展，就可以对我们的项目进行构建。（参考链接）

1.1.1. 关于 task

一个 Project 包含多个 task，一个 Task 就是一个原子性的操作，比如一次编译或一次打包。Android Gradle 插件基本包含了所有的 Java 插件功能，比如 assemble、check、build 等，此外，还添加了 connectedCheck、deviceCheck、lint、install 和 uninstall 等任务。一般我们常用的任务是 build，assemble, clan，lint，check 等，这些任务可以打包生成 apk。

1.1.2. 关于插件

1.1.2.1.插件功能

阿斯蒂芬

1.1.2.2.引入插件

引入插件是通过 Project.apply()方式完成的，apply 方法有好几种用法，且插件分为二进制和脚本插件。二进制插件应用方式如代码 1 所示。

```
apply plugin: 'com.google.gms.google-services' // 1)
apply plugin: 'com.android.application'      // 2)
apply plugin: org.gradle.api.plugins.JavaPlugin // 3)
apply plugin: JavaPlugin          // 4)
apply plugin: 'java'              // 5)
apply from: 'version.gradle' // 6)
apply{                            // 7)
    plugin 'java'
}
```

代码 1

在代码 1 中，第 5 行中的'java'是 Java 插件的 plugin id，其对应的类型是第 3 行，包 ort.gradle.api.plugins 是默认导入的，所以第 3、4、5、7 行代码等价，是引入 Java 插件的四种写法，且都是二进制引入。第 6 行代码是脚本插件的引入方式，其余都是二进制引入方式。

应用第三方插件，必须在 buildscript{}里配置其 classpath 才能使用，比如 Android Gradle 插件就属于 Android 发布的第三方插件，可以在项目工程根目录下的 build.gradle 里面配置（为各个子工程共有，如此，子工程便不需各自重复配置），也可以在各个子工程里面配置，如代码 2 所示。

```
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath
'com.android.tools.build:gradle:3.2.0-alpha08'
        classpath 'com.google.gms:google-services:3.2.0'
其余代码省略。。。
```

代码 2

在代码 2 中，buildscript{}块是一个在构建项目之前，为项目进行前期准备和初始化相关配置依赖的地方，配置好所需的依赖，就可以应用插件了（见代码块 1，第 2 行）。classpath 的内容格式为：group:name:version，即用冒号隔开。

1.1.3. Java Gradle 插件简介

爱迪生发

1.2. Android Gradle 简介

Android 插件基于 Java 插件，即包含了 Java 插件的所有功能。配置块 android{}就是 Android Gradle 插件为 Project 对象添加的一个扩展，所有的 Android Gradle 工程的配置都在 android{}这个唯一入口中。每个项目至少有 2 个 build.gradle 文件，一个是项目级的，另一个是模块级的。

Android 工程 Gradle 配置文件包含如下：

名称	简介

build.gradle(工程级)	位于项目根目录下，于定义适用于项目中所有模块的构建配置。
	例如指定 Gradle 插件版本：
	`classpath 'com.android.tools.build:gradle:3.2.0'`
	配置各个子模块共用版本号：
	`project.ext {`
	`  MINIMUM_SDK_VERSION = 14`
	`  TARGET_SDK_VERSION = 27`
	`  TOOLS_VERSION = "27.0.3"`
	`  SUPPORT_VERSION = "27.0.2"`
	`}`

Build.gradle（模块级）	
Gradle.properties	Gradle 运行环境配置文件。可配置 java 运行环境： `org.gradle.jvmargs=-Xmx1536m`
Gradle-wrapper.properties	
Local.properties	为构建系统配置本地环境属性，例如 SDK 安装路径。（不应手动修改该文件，不需要添加到 git 版本库） 也可配置一些本地化变量（后续详细讲解）。
Progard-rules.pro	定义混淆规则
Setting.gradle	位于项目根目录下，用于指示 gradle 在 build 应用的时候需要将哪些模块包括在内。例如：include ':app'。
Gradlew.bat	自动完成 gradle 的 Windows 脚本

2. 关于 GradleWrapper 和 gradle 版本

Gradle 版本问题导致无法编译时常发生，在配置版本的时候，需要对照 gradle 和 Gradle 插件对应表，如下所示：

Plugin version	Required Gradle version
1.0.0 - 1.1.3	2.2.1 - 2.3
1.2.0 - 1.3.1	2.2.1 - 2.9
1.5.0	2.2.1 - 2.13
2.0.0 - 2.1.2	2.10 - 2.13
2.1.3 - 2.2.3	2.14.1+
2.3.0+	3.3+
3.0.0+	4.1+
3.1.0+	4.4+
3.2.0 - 3.2.1	4.6+
3.3.0 - 3.3.2	4.10.1+

例如设置 gradle 版本为 4.6，设置插件版本为 3.2.0：

```
distributionUrl=https\://services.gradle.org/distributions/gradle-4.6-all.zip
```

```
classpath 'com.android.tools.build:gradle:3.2.0'
```

2.1. 其他

3. 常见配置

3.1. 常用配置

属性	简介
BuildToolVersion	Android 构建工具的版本。可以在 Android SDK 目录里看到，是一个工具包，包括 appt，dex 等工具。 Android Gradle 插件各个版本各自有一个最低 Android SDK Build Tool 的版本要求。
compileSdkVersion	不会影响运行时行为，建议用最新 sdk 编译。必须大于 support library 的版本号。
targetSdkVersion	决定了系统 api 的行为，即只要 targetSdkVersion 保持不变，在任何版本的系统上，行为都不会改变。
minSdkVersion	App 最低的系统版本要求，大于等于所依赖库的 minSdkVersion。

4. 关于 Local.properties

此文件在 Android Studio 中是用来配置 SDK 目录的,也可以在文件中配置一些本地化的变量。这个文件不会被提交到版本控制中。为了安全起见签名文件不可以放在项目中，那么可能每个开发人员的存放地方不同，这种情况下就可以使用 local.properties 来配置路径然后在 app 目录下的 build.gradle 中引用，示例如下：

在 local.properties 配置如下，其中 sd.dir 是自动生成的，无需改动：

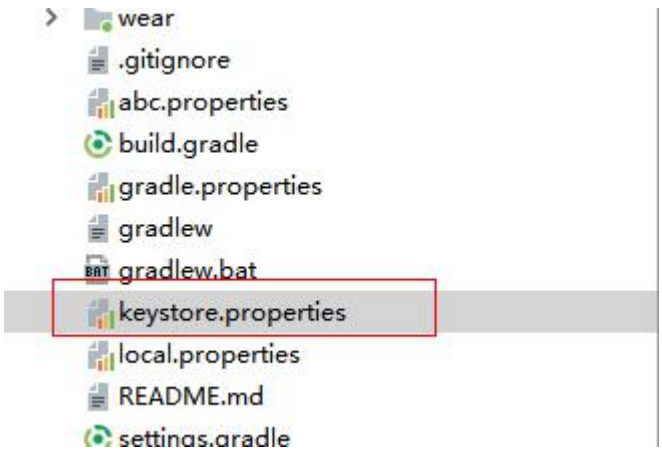
```
sdk.dir=C\:\\Users\\Administrator\\AppData\\Local\\Android\\Sdk
RELEASE_KEY_FILE_PATH=E\:\\
```

在 build.gradle 中配置如下，通过 java 的 Properties 对象读取 local.properties 的键值映射内容：

```
signingConfigs{
    autoSigning{
        def properties = new Properties()
        def inputStream =
project.rootProject.file("local.properties").newDataInputStream();
        properties.load(inputStream)

        keyAlias 'myAlias'
        keyPassword 'myPwd'
        storeFile file(properties.getProperty('RELEASE_KEY_FILE_PATH'))
        storePassword 'myStorePwd'
    }
}
```

亦或新建一个 keystore.properties 文件如下：



在其中定义一些本地配置：

```
storeFile=keystore.jks
storePassword=myStorePassword
keyAlias=myKeyAlias
keyPassword=mykeyPassword
```

5. 关于 defaultConfig

属性如下：

6. 关于 ProductFlavor

属性如下：

属性	简介
dimension	

6.1. defaultConfig

DefaultConfig 是一个 ProductFlavor，负责定义所有的默认配置，如果一个 ProductFlavor 没有被特殊定义配置的话，默认就会使用 defaultConfig{}块指定的配置，比如包名、版本号、版本名称等，属性如下：

属性	简介
applicationId	
versionCode	表明 app 内部版本号，用于控制 app 升级。
versionName	表明 app 应用的版本名称。
minSdkVersion	指定 app 最低支持的 Android 操作系统版本
targetSdkVersion	
SigingConfig	配置默认签名信息
ProguardFile	配置混淆文件

6.2. 多 apk 生成

Android Gradle 支持基于屏幕分辨率、Android 版本、设备类型（平板或手机）、设备硬件（是否有摄像头）等生成多个 apk 文件，而发布的时候这些 apk 共享一个应用 ID 和下载入口，如此可以缩小 apk 大小。

7. 关于 apk 自定义命名

有时候会遇到“The APK file does not exist on disk”的错误，导致 apk 无法安装。原因在于命名规则中包含动态部分，由于 Gradle 在执行编译命令和安装命令有时间差，导致编译出来的 apk 名称和安装时获取到的 apk 名称不一致，所以它就报找不到指定的 apk 文件了。那么应该这样命名呢？如代码 3 所示（Gradle 插件 3.2 以下略有不同）。

```
android.applicationVariants.all { variant ->
    variant.outputs.each {
            if (variant.buildType.name.equals("release")) {
                variant.outputs.all {
                    def application = "PowerSecurity"
                    def versionCode = variant.getVersionCode()
                    def versionName = variant.getVersionName()
                    def channel =
variant.productFlavors[0].manifestPlaceholders.get("CHANNEL_VALUE")
                    def newName =
"${application}_${versionName}_${versionCode}_${channel}.apk"
                    outputFileName = newName;
//                }
                }
            }
    }
```

代码 3 列出了导出的 aar 包的命名方法，需要注意的是命名代码块必须放在 android{}下面，不能放在 buildTypes{}或其他子配置里面。如果需要对 app 包命名则把”libraryVariants”改为 applicationVariants 即可。

7.1. 多渠道打包

Android Gradle 中，定义了一个叫做 Build Variant 的概念，一个 Build Variant = Build Type（Release 或 Debug） + Product Flavor（各种渠道，例如 google、badu 等），参考链接， 参考链接，

7.2. 动态引入组件

参考代码如下：

```
def allComponents = ["app", "facebook"]
allComponents.forEach(
        { name ->
            if (name.equals("")){
                include ":${name}"
                project(":${name}").projectDir == getComponentDir(name)
            }
        }
)
```

8. 关于依赖

依赖分为三种类型模块依赖、远程二进制依赖、本地二进制依赖。查看所有依赖命令：
Gradlew androidDependencies
查看某个子模块（例如 app-dir）的依赖：
Gradlew :app-dir:denpendencies
模块依赖，例如 compile project(':mylibrary')；远程二进制依赖，例如 compile 'com.android.support:appcompat-v7:27.1.1'；本地二进制依赖，例如 compile fileTree(dir: 'libs', include: ['*.jar'])，如下：

```
repositories {
    flatDir {
        dirs 'libs' // 本地库目录，'以 build.gradle 所在目录为根目录的相对路径'
    }
    maven { url "https://s3.amazonaws.com/moat-sdk-builds" }
    ivy{
        url "xxx"
    }
    maven { url 'https://maven.google.com' }
}
```

8.1. 动态添加依赖

经常由于所依赖的版本不一致，发生各种编译或者运行时错误，以 com.android.support 库为例，有两个办法，办法一如下所示，在 app 模块的 build.config 的最外层添加如下代码，指定所有版本为 26.0.0：

```
configurations.all {
    resolutionStrategy {
        eachDependency { details ->
            // Force all of the primary support libraries to use the same
version.
            if (details.requested.group == 'com.android.support') {
                details.useVersion "26.0.0"
            }
        }
    }
```

指定版本为 2.1.2。

```
configurations.all {
  resolutionStrategy {
    eachDependency { details ->
      // Force all the error-prone dependencies to use the same version.
      if (details.requested.group == 'com.google.errorprone' &&
              details.requested.name.startsWith('error_prone_')) {
        details.useVersion '2.1.2'
      }
    }
  }
}
```

办法二如下，在添加依赖的时候分别添加版本号变量：

```
compile "com.android.support:percent:$support_version"
```

其中 support_version 变量定义在工程的 build.config 中 buildscript 的最顶层：

```
buildscript {
    ext.support_version = '27.1.0'
    repositories {
省略...
```

参考代码如下：

```
boolean useSource = gradle.startParameter.taskNames.any {
    it.contains("assemble") || it.contains("install") || it.contains("upload")
}
subProject.forEach { subProject ->
    if (useSource) {
        android.sourceSets.main {
            java.srcDirs += file("../${subProject}/src/main/java")
            res.srcDirs += file("../${subProject}/src/main/res")
        }
    } else {
        dependencies { implementation project(":$subProject") }
    }
}
```

9. SourceSet

sourceSet 可以为不同的编译包指定不同的源文件（包括 manifest、java、res 等）。示例如下指定不同的 manifest 文件：

```
sourceSets{
    main{
        if (isLibrary.toBoolean()){
            manifest.srcFile "src/main/AndroidManifest.xml"
        }else {
            manifest.srcFile "src/main/AndroidManifest.xml"
        }
    }
    google{
        manifest.srcFile "src/google/AndroidManifest.xml"
    }
    baidu{
        manifest.srcFile "src/baidu/AndroidManifest.xml"
    }
}
flavorDimensions "channel"
productFlavors{
    google{
        dimension "channel"
    }
    baidu{
        dimension "channel"
    }
}
```

10.其他

10.1. BuildConfig

Android Gradle 提供了 buildConfigField(String type, String name, String value)让我们可以添加自己定义的常量到 BuildConfig 中，可以为不同的渠道或者不同构建类型在 BuildConfig 中定义不同的值。

10.2. applicationId 和 package 属性值的关系

App 模块 Build.gradle 中的 applicationId 是指应用的 id，package 仅仅只是资源路径和源码目录。

11.关于 Build

11.1. 关于 build 慢的问题

11.1.1. 加快 DEX 生成速度

Android Studio 提供 dexOption 区块以便于我们配置 DEX 构建属性，加快 DEX 文件的生成速度，代码如下：

```
dexOptions {
 preDexLibraries true
  maxProcessCount 8
  javaMaxHeapSize "2048m"
}
```

preDexLibraries 声明是否预先编译依赖库，从而加快构建速度，实质是通过延时清除已生成的依赖库的构建文件，从而提高构建速度，根据使用情况合理配置。maxProcessCount 设置进程运行过程中可以使用的最大线程数。默认值为 4。javaMaxHeapSize 设置 DEX 编译器的最大堆大小，堆或者栈都是用于存放暂时不用的垃圾，当内存不足时，垃圾回收机制会清除过时的缓存，堆大小决定垃圾清除的频率，影响着构建的速度。设置如下：

```
org.gradle.jvmargs=-Xmx4608M
```

11.2. 参考 Build 步骤

建议用 gradlew 命名完成整个 build 任务（也可配置 gradle-2.14.1 的环境变量）。

11.3. 解决依赖库版本冲突

强制设置依赖库版本：

```
configurations.all{
    resolveStrategy{
        force 'org.hamcrest:hamcrest-core:1.3'
    }
}
```

查看依赖库命令：gradlew -q app:dependencies

12.常见错误与参考解决办法

12.1. 找不到原因的错误

有时候会遇到很奇怪的错误提示，或者没有提示（Run Configurations 里面找不到 Module），导致当前工程无法完成编译。可以尝试的解决办法是手动删除当前的 build 下的编译文件，随后执行“Invalid cache/Restart”，还是无法编译则修改 gradle 和 gradle-wrapper 版本。

12.2. 同步失败

错误信息：

```
Unfortunately you can't have non-Gradle Java modules and Android-Gradle modules in one project
```

建议尝试以下方式解决：

12.3. Java 文件出现棕红色时钟，文件无法跳转

在打开工程的时候，停在 Indexing 过程中，文件无法跳转，并出现如下提示：

```
Argument for @NotNull parameter 'message' of com/android/tools/idea/gradle/project/sync/GradleSyncState.syncFailed must not be null
```

建议尝试以下方式解决此问题：

- 点击 File->sync Project width gradle Files 和 File->Sync width File System.
- 或者点击 File->invalidate Cache/System.
- 或者重命名当前工程文件的父目录，然后用 Android studio 再次打开此工程，或许能解决此问题。

12.4. 禁止使用动态版本依赖

相关网址链接。动态版本依赖可能会导致库的行为不可控（即会出现各种奇奇怪怪的编译或运行错误），尽量用如下形式依赖，确保依赖版本的统一性和确定性：

```
api "com.android.support:appcompat-v7:$project.SUPPORT_VERSION"
```

12.5. 找不到库文件

添加仓库或者更换翻墙工具可以解决问题。

12.6. 奇怪错误 1

错误提示：<item name.....
解决办法：删除 manifest 里面的所有注释。

12.7. BuildTools 版本问题

错误日志如下：

```
E/AndroidRuntime: FATAL EXCEPTION: main
    java.lang.NoSuchMethodError: No static method
getFont(Landroid/content/Context;ILandroid/util/TypedValue;ILandroid/widget/Text
View;)Landroid/graphics/Typeface; in class
Landroid/support/v4/content/res/ResourcesCompat; or its super classes (declaration
of 'android.support.v4.content.res.ResourcesCompat' appears in
```

解决办法：
把 compileSdkVersion 和 buildToolsVersion 改为和所有的“com.android.support:*”版本改为一致。即如果 support 版本为 27，那么 compileSdkVersion 和 buildToolsVersion 版本都必须为 27。

12.8. 依赖库版本不一致问题

提示错误：

Duplicate zip entry
[classes.jar:android/support/design/widget/CoordinatorLayout$Behavior.class]

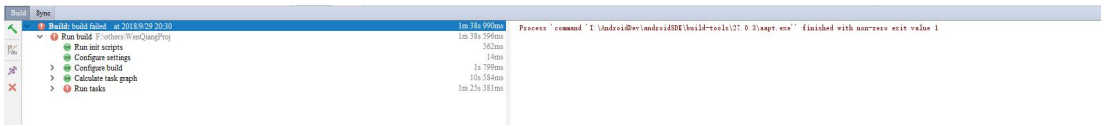
解决步骤:

- 运行命令查看依赖版本库：gradlew -q :app:dependencies > dependencies.txt
- 用 ctrl+F 查找 dependencies.txt 里面的各个 support 版本信息情况。
- 把所有的 support 版本改为->指向的版本（参考 8.1），且按照 12.7 的要求把 BuildToolsVersion 改为一致，就能编译通过了。

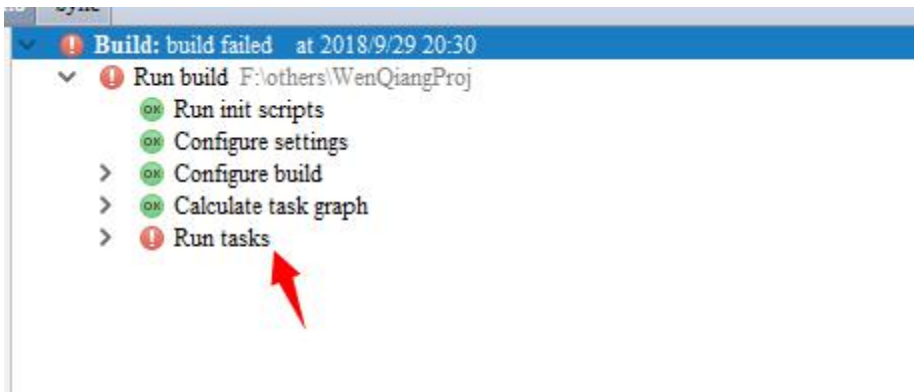
12.9. appt 错误

有时候，突然出现 appt 错误（资源文件错误），错误详情，如下：

Process 'command 'I:\AndroidDev\androidSDK\build-tools\27.0.3\aapt.exe'' finished with non-zero exit value 1



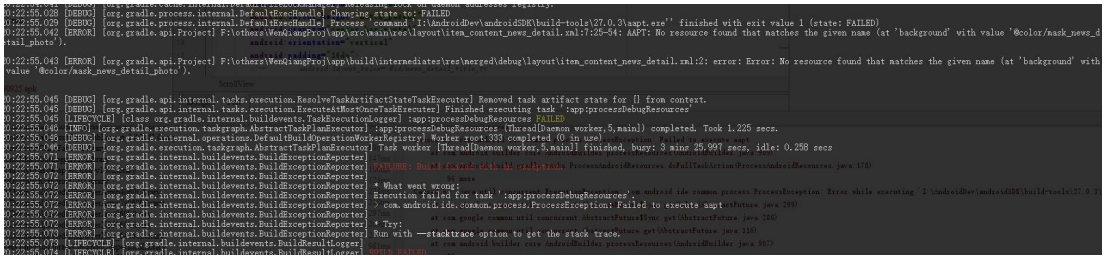
出现此问题有很多原因，参考解决过程如下：
点击如下红色警告：



展开如下所示：

可知道是在执行任务 processDebugResources 出现的错误，在控制台下，转到项目的根目录，执行如下命令：

得到错误的详细信息如下所示：



根据上图错误提示，修改即可。

附录 1

参考网：

1、Android Plugin DSL Reference：http://google.github.io/android-gradle-dsl/current/index.html

2、ProductFlavor：

http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html

3、gradle.properties：https://docs.gradle.org/current/userguide/build_environment.html

4、中文官方：http://ask.android-studio.org/?/article/7

参考书籍：

1、Android Gradle 权威指南.pdf